
Microsoft®
PDC 2000
Professional Developers Conference
the defining point

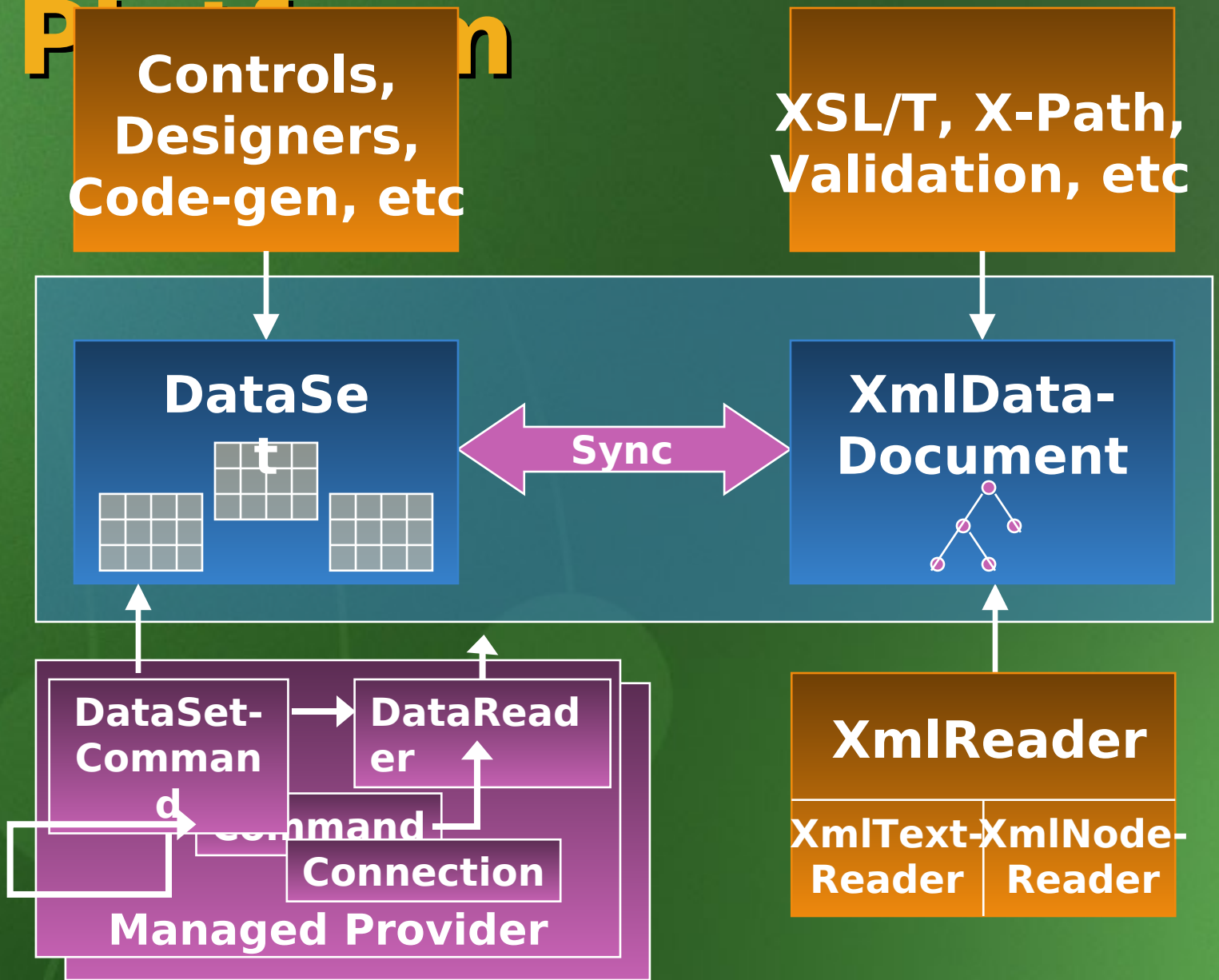
Using ADO+

Michael Pizzo/Tom Kaiser
Program Management
WebData/Frameworks

Agenda:

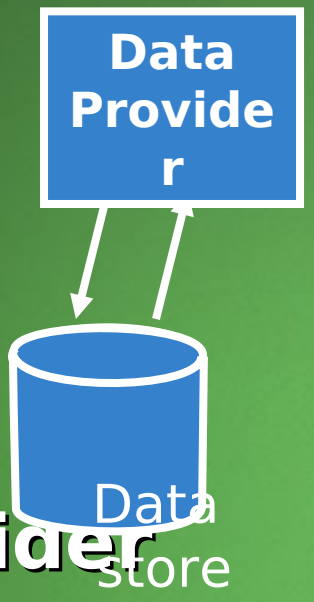
- ADO+ Architecture
  - ADO+ Object Model
- DataBinding
- ADO+ and XML
- Using ADO+ from the Designers
- Summary

Data in the Web Platform



Managed Providers

- Manage interaction to a data source
- Managed equivalent of OLE DB layer
  - Directly exposes consumer interfaces
    - No more COM/Automation dichotomy
- Microsoft Implementations
  - ADO Managed Provider
    - Access to any OLE DB Provider
  - SQL Server Managed Provider



ADO+ Object Model

- Classic ADO Styling…
  - Connection
  - Command, Parameter
- …but not your father's Oldsmobile
  - DataReader
    - Forward-only, Read-only "RecordSet"
  - DataSet
    - Disconnected, In-Memory Cache
  - DataSetCommand

Connection Object

- Represents a connection to the data source
- On a Connection you can…
  - Customize the connection to the database
  - Begin, commit, and abort transactions
- Equivalent to the ADODB.Connection object

Connection Object

```
//Specify the System.Data.ADO Namespace
Using System.Data.ADO;

// Create an instance of an ADOConnection object
ADOConnection cnn = new ADOConnection();

// Set the connection string
cnn.ConnectionString = "provider=SQLOLEDB;
       datasource={local};user id=sa;database=pubs";

//Open the Connection
cnn.Open();
```

Command Object

- Represents a command to be executed
  - Not necessarily SQL
- With an ADO command you can:
  - Define a statement to be executed on the server
  - Set parameter information for that command
  - Retrieve return values from command execution
- Corresponds to ADODB.Command object
- May contain parameters

Command Object

```
// Create Command
ADOCommand cmd = new ADOCommand();

// Set command's active connection and command text
cmd.ActiveConnection = cnn;
cmd.CommandText =
    "Select au_lname from authors where state = ?";

// Create parameter and set value
cmd.Parameters.Add( "param1", ADODBType.Char,2);
cmd.Parameters["param1"].Value = "CA";
```

DataReader

- The DataReader provides a forward-only, read-only stream over the data
  - Represents results of an executed query/command
- The DataReader enables you to...
  - Obtain a stream of results from a data source
- Equivalent to a FO/RO RecordSet
  - Doesn't support scrolling, updating
  - Fields accessed through strongly typed accessors, rather than FieldsCollection
    - Performance
    - Still supports myRow["fieldname"] access
- Supports DataBinding (in Beta 1)

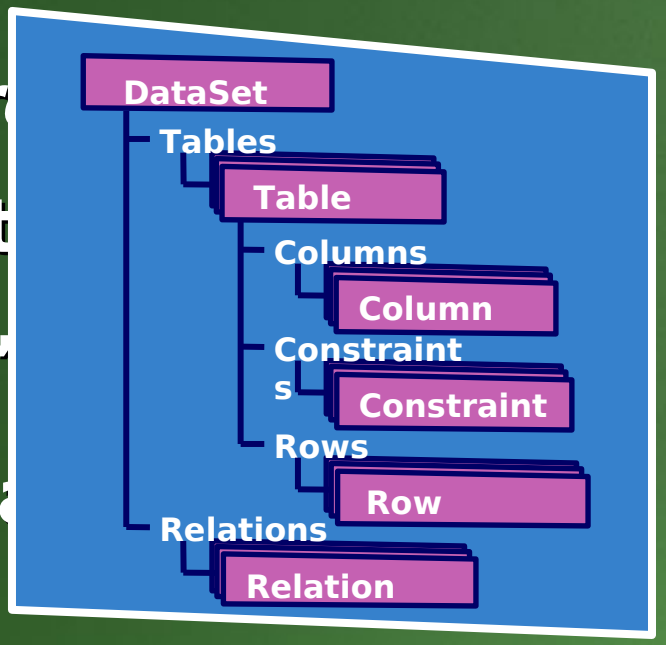
Getting Data

```
// Define DataReader
IDataReader dr;

// Execute Command and retrieve results
cmd.Execute(out dr);
while(dr.Read())
    {
    Console.WriteLine("Name:" + dr["au_lname"]);
    }
dr.Close();
```

DataSet

Common client data stor



- Relational View of Dat
  - Tables, Columns, Rows, Constraints, Relations
- Directly create metad and insert data
- Explicit Cache Model
  - Disconnected, remotable object
  - No knowledge of data source or properties
    - Common Behavior
    - Predictable performance characteristics
  - Array-like indexing

DataSet

```
// Create a "Pubs" DataSet
DataSet pubs = new DataSet("Pubs");

//Create an "Inventory" Table
DataTable inventory = new DataTable("Inventory");
inventory.Columns.Add("TitleID", typeof(Int32));
inventory.Columns.Add("Quantity", typeof(Int32));

// Add Inventory table to Pubs DataSet
pubs.Tables.Add(inventory);

// Add a record to the Inventory table
DataRow row = inventory.NewRow();
row["TitleID"]=1;
row["Quantity"]=25;
inventory.Rows.Add(row);
```

Strongly Typed DataSet

- Deal with DataSets, Tables, Rows as Objects
  - Columns, Relations as properties

```
//print out each author and their titles
  foreach (Author myAuthor in Pubs.Authors.Rows)
  {
    Console.WriteLine("Name = " +
myAuthor.au_lname);
    foreach (Title myTitle in myAuthor.Titles)
    {
      Console.WriteLine("Title = " +
myAuthor.Title);
    }
  }
```

DataSetCommand

- Knows how to load a table from a data store and write changes back.
  - Exposes two methods:
    - FillDataSet(DataSet)
    - Update(DataSet)
  - Provides mappings between tables & columns
  - User can override default insert/update/delete commands
    - i.e., to specify stored procedures

DataSetCommand

```
// Create a DataSetCommand
ADODataSetCommand dataSetCommand = new ADODataSetCommand(
            "Select * from authors", cnn);
dataSetCommand.FillDataSet(pubs, "Authors");

// make some changes to the author data in the dataset
pubs.Tables[0].Rows[0]["state"]="WA";

// submit changes back to datasource
dataSetCommand.Update(pubs, "Authors");
```

DataBinding

- DataView
  - Thinks of this as a view on the DataTable
  - Allows setting Sort Order and Filter on a view of the table
  - Any number of DataViews can be created on a table to enable different views of the same table
  - Used for databinding
- DataSetView
  - Think of this as a view on top of the DataSet
  - Allows setting sort orders and filters
  - Allows “linking” of DataViews

DataBinding

- Sources for DataBinding
  - DataReader
  - DataTable
  - DataView
  - DataSet
  - DatSetView
  - Array
  - Collection
  - IList

ADO+ and XML

- The DataSet
  - Loads/saves XML data into/out of DataSet
  - Schema can be loaded/saved as XSD
  - Schema can be inferred from XML Data
- The DataSet can be associated with an XmlDataDocument
  - Exposes a relational view over structured XML
    - According to the DataSet schema
  - Allows strong typing, control binding, relational access of XML data
  - Preserves full fidelity of XML Document
  - Simultaneously exposes data relationally or as XML
    - Semi-structured data

ADO+ and XML

```
// Associate an XmlDataDocument with the DataSet
XmlDataDocument xmlDocument = new XmlDataDocument(pubs);

// Get an XmlNavigator for the XmlDataDocument
DataDocumentNavigator xmlNavigator = new
            DataDocumentNavigator(xmlDocument);

// Select all the authors from CA
xmlNavigator.Select("Authors[@state='CA']");

// Write out all of the authors' last names
while(xmlNavigator.MoveToNextSelected())
{
    Console.WriteLine("Name = " +
                  xmlNavigator.GetAttribute("au_lname"));
}
```

Summary

- ADO+ is a natural evolution of classic ADO
  - Common Connection/Command model
  - Separate persistence model from programming model
    - Optimized ForwardOnly/ReadOnly Result Stream
    - Explicit, disconnected relational cache
- ADO+ is designed to work with XML

Related Sessions

- Other ADO+ and XML Framework talks
  - ADO and XML Overview – Omri Gazitt
  - Using the XML Framework – Chris Lovett
  - XML in Action – William Adams
  - Using XML and Data in ASP+ – Mike Pizzo
  - Data and XML in VS7 – Sean Draine
- Other WebData talks

Where do you want to go today?

Microsoft